



Definition

Refactoring (Verb):

Umstrukturierung von Software durch Anwendung einer Reihe von Refactorings, ohne das beobachtbare Verhalten der Software zu ändern.

Refactoring (Substantiv):

eine Änderung der internen Struktur einer Software, um sie verständlicher und einfacher modifizierbar zu machen, ohne das beobachtbare Verhalten zu verändern.

— Martin Fowler

"Refactoring" hat ernsthaften Schaden genommen, als die Leute begannen, damit lange Pausen bei der Feature-Entwicklung zu erklären.

Dabei verschwand sogar der Teil mit dem »Verhalten beibehalten«, sodass ein Refactoring das System problemlos aus dem Tritt bringen kann. Keine neuen Features, möglicher Schaden und nichts, was man am Ende vorzeigen kann. Danke, ich verzichte.

— Kent Beck



REFACTORING

The image depicts a dark, desolate landscape under a heavy, black, smoke-filled sky. In the center, the word "REFACTORING" is rendered in large, bold, capital letters. Each letter is filled with a bright, intense fire, with flames and embers appearing to rise from the characters. The ground is a dark, cracked expanse of what appears to be cooled lava, with several streams of molten lava flowing across it, glowing with orange and yellow light. In the background, the silhouettes of skeletal, leafless trees are visible against the dark sky. The overall atmosphere is one of intense heat, destruction, and a sense of a world in flames.

Aufräumereien sind eine Untermenge der Refactorings. Sie sind niedliche, knuddelige kleine Refactorings, die niemand wirklich hassen kann.

— Kent Beck





Tidying!

▼ Part I. Tidying

Chapter 1. Guard Clauses

Chapter 2. Dead Code

Chapter 3. Normalize Symmetries

Chapter 4. New Interface, Old Implementation

Chapter 5. Reading Order

Chapter 6. Cohesion Order

Chapter 7. Move Declaration and Initialization Together

Chapter 8. Explaining Variables

Chapter 9. Explaining Constants

Chapter 10. Explicit Parameters

Chapter 11. Chunk Statements

Chapter 12. Extract Helper

Chapter 13. One Pile

Chapter 14. Explaining Comments

Chapter 15. Delete Redundant Comments

Guard Clauses

```
1 function calculateDiscount(price: number, isHoliday: boolean, isVIP: boolean) {  
2     let discount = 0;  
3     if (isHoliday) {  
4         if (!isVIP) {  
5             discount = price * 0.1;  
6         }  
7     }  
8     return discount;  
9 }
```

Guard Clauses

```
function calculateDiscount(price: number, isHoliday: boolean, isVIP: boolean) {  
  let discount = 0;  
  if (isHoliday) {  
    if (!isVIP) {  
      discount = price * 0.1;  
    }  
  }  
  return discount;  
}
```



```
function calculateDiscount(price: number, isHoliday: boolean, isVIP: boolean) {  
  if (!isHoliday) return 0;  
  if (isVIP) return 0;  
  
  return price * 0.1;  
}
```

Dead Code

```
1 class BankAccount {
2     private balance: number = 0;
3     private transactionHistory: number[] = [];
4
5     private calculateAverageTransaction(): number {
6         const sum = this.transactionHistory.reduce(
7             (a, b) => a + b, 0);
8         return sum / this.transactionHistory.length;
9     }
10
11    public getBalance(): number {
12        return this.balance;
13    }
14
15    public printStatementFormA(): void {
16        console.log("Balance: ", this.getBalance());
17        console.log("Average Transaction: ",
18            this.calculateAverageTransaction());
19    }
20
21    public printStatementFormB(): void {
22        console.log("Balance: ", this.getBalance());
23        console.log(account.getTransactionHistory());
24    }
```

```
    public deposit(amount: number): void {
        this.balance += amount;
        this.transactionHistory.push(amount);
    }

    public getTransactionHistory(): number[] {
        return this.transactionHistory;
    }

    public withdraw(amount: number): void {
        if (amount <= this.balance) {
            this.balance -= amount;
            this.transactionHistory.push(-amount);
        } else {
            console.log("Insufficient balance");
        }
    }
}

const specialTransactionPrint =
    (inValueFromT1, inValueFromT2, outValueFromT3) => {
        const account = new BankAccount();
        account.deposit(inValueFromT1);
        account.deposit(inValueFromT2);
        account.withdraw(outValueFromT3);
        printStatementFormB();
    }
```


Normalize Symmetries

```
1  public class Person {
2  ...
3      // Using string concatenation
4      public String getFullName() {
5          return firstName + " " + lastName;
6      }
7
8      // Using String.format
9      public String getLocation() {
10         return String.format("%s, %s", city, country);
11     }
12
13     // Using StringBuilder
14     public String getDetailedInfo() {
15         return new StringBuilder(firstName).append(" ").append(lastName)
16             .append(" from ").append(city).append(", ").append(country).toString();
17     }
18 }
```

```
// Using string concatenation
public String getFullName() {
    return firstName + " " + lastName;
}
// Using String.format
public String getLocation() {
    return String.format("%s, %s", city, country);
}
// Using StringBuilder
public String getDetailedInfo() {
    return new StringBuilder(firstName).append(" ").append(lastName)
        .append(" from ").append(city).append(", ").append(country).toString();
}
```



```
public String getFullName() {
    return firstName + " " + lastName;
}

public String getLocation() {
    return city + ", " + country;
}

public String getDetailedInfo() {
    return firstName + " " + lastName + " from " + city + ", " + country;
}
```

New Interface, Old Implementation

```
class Database:
    def connect(self, url, username, password):
        ...

    def execute_query(self, query):
        ...

    def disconnect(self):
        ...
```



```
def connect(self, url, username, password):
def execute_query(self, query):
def disconnect(self):

def execute_query(self, url, username, password, query):
    self.connect(url, username, password)
    self.execute_query(query)
    self.disconnect()
```

Usage

```
db = Database()
db.execute_query("url", "username", "password", "SELECT * FROM table")
```

Reading Order

```
1 class Car:
2     def start_engine(self):
3         if self.check_battery():
4             print("Engine started.")
5         else:
6             print("Engine failed to start. Battery is dead.")
7
8     def __init__(self, make, model, year, color):
9         self.make = make
10        self.model = model
11        self.year = year
12        self.color = color
13
14    def check_battery(self):
15        # For simplicity, let's assume the battery is good if the car is less than 5 years old
16        return datetime.datetime.now().year - self.year < 5
```

```
def start_engine(self):
    if self.check_battery():
        print("Engine started.")
    else:
        print("Engine failed to start. Battery is dead.")

def __init__(self, make, model, year, color):
    ...

def check_battery(self):
    return datetime.datetime.now().year - self.year < 5
```



```
def __init__(self, make, model, year, color):
    ...

def check_battery(self):
    # For simplicity, let's assume the battery is good if the car is less than 5 years old
    return datetime.datetime.now().year - self.year < 5

def start_engine(self):
    if self.check_battery():
        print("Engine started.")
    else:
        print("Engine failed to start. Battery is dead.")
```

Cohesion Order

```
1 class ShoppingCart:
2     def __init__(self):
3         self.items = []
4
5     def apply_discount(self, code):
6         if self.is_valid_discount_code(code):
7             total = self.calculate_total()
8             print(f"Total after discount: {total * 0.9}")
9         else:
10            print("Invalid discount code.")
11
12    def calculate_item_total(self, item):
13        return item['price'] * item['quantity']
14
15    def add_item(self, name, price, quantity):
16        self.items.append({'name': name, 'price': price, 'quantity': quantity})
17
18    def calculate_total(self):
19        ...
20
21    def is_valid_discount_code(self, code):
22        return code.startswith("DISC")
```

Cohesion Order

```
def __init__(self): ...  
  
def apply_discount(self, code): ...  
  
def calculate_item_total(self, item): ...  
  
def add_item(self, name, price, quantity): ...  
  
def calculate_total(self): ...  
  
def is_valid_discount_code(self, code): ...
```



```
def __init__(self): ...  
  
def add_item(self, name, price, quantity): ...  
  
def calculate_item_total(self, item): ...  
  
def calculate_total(self): ...  
  
def is_valid_discount_code(self, code): ..  
  
def apply_discount(self, code): ...
```

Move Declaration and Initialization Together

```
1 let total: number;
2 // ...some code that doesn't use total
3 // ...some code that doesn't use total
4 // ...some code that doesn't use total
5 total = 0;
6 for (let item of items) {
7     let price: number;
8     // ...some more code, maybe it uses total but doesn't use price
9     // ...some more code, maybe it uses total but doesn't use price
10    // ...some more code, maybe it uses total but doesn't use price
11    price = item.price;
12    total += price * item.quantity;
13 }
14 return total;
```


Move Declaration and Initialization Together

```
let total: number;
// ...some code that doesn't use total
total = 0;
for (let item of items) {
  let price: number;
  // ...some more code, maybe it uses total but doesn't use price
  price = item.price;
  total += price * item.quantity;
}
return total;
```



```
let total = 0;
// ...some code that doesn't use total
for (let item of items) {
  let price: number;
  // ...some more code, maybe it uses total but doesn't use price
  price = item.price;
  total += price * item.quantity;
}
return total;
```

Explaining Variables

```
1 def calculate_point():  
2     return (math.sqrt(16) * math.sin(math.radians(45)), math.sqrt(16) * math.cos(math.radians(45)))
```

Explaining Variables

```
def calculate_point():  
    return (math.sqrt(16) * math.sin(math.radians(45)), math.sqrt(16) * math.cos(math.radians(45)))
```



```
def calculate_point():  
    radius = math.sqrt(16)  
    angle_in_radians = math.radians(45)  
  
    x = radius * math.sin(angle_in_radians)  
    y = radius * math.cos(angle_in_radians)  
  
    return (x, y)
```

Explaining Constants

```
1 function handleResponse(response: {code: number, data: any}): void {
2     if (response.code === 404) {
3         console.log("Page not found.");
4     } else if (response.code === 200) {
5         console.log("Success!");
6     }
7 }
```

Explaining Constants

```
function handleResponse(response: {code: number, data: any}): void {  
  if (response.code === 404) {  
    console.log("Page not found.");  
  } else if (response.code === 200) {  
    console.log("Success!");  
  }  
}
```



```
const PAGE_NOT_FOUND = 404;  
const SUCCESS = 200;  
  
function handleResponse(response: {code: number, data: any}): void {  
  if (response.code === PAGE_NOT_FOUND) {  
    console.log("Page not found.");  
  } else if (response.code === SUCCESS) {  
    console.log("Success!");  
  }  
}
```

Explicit Parameters

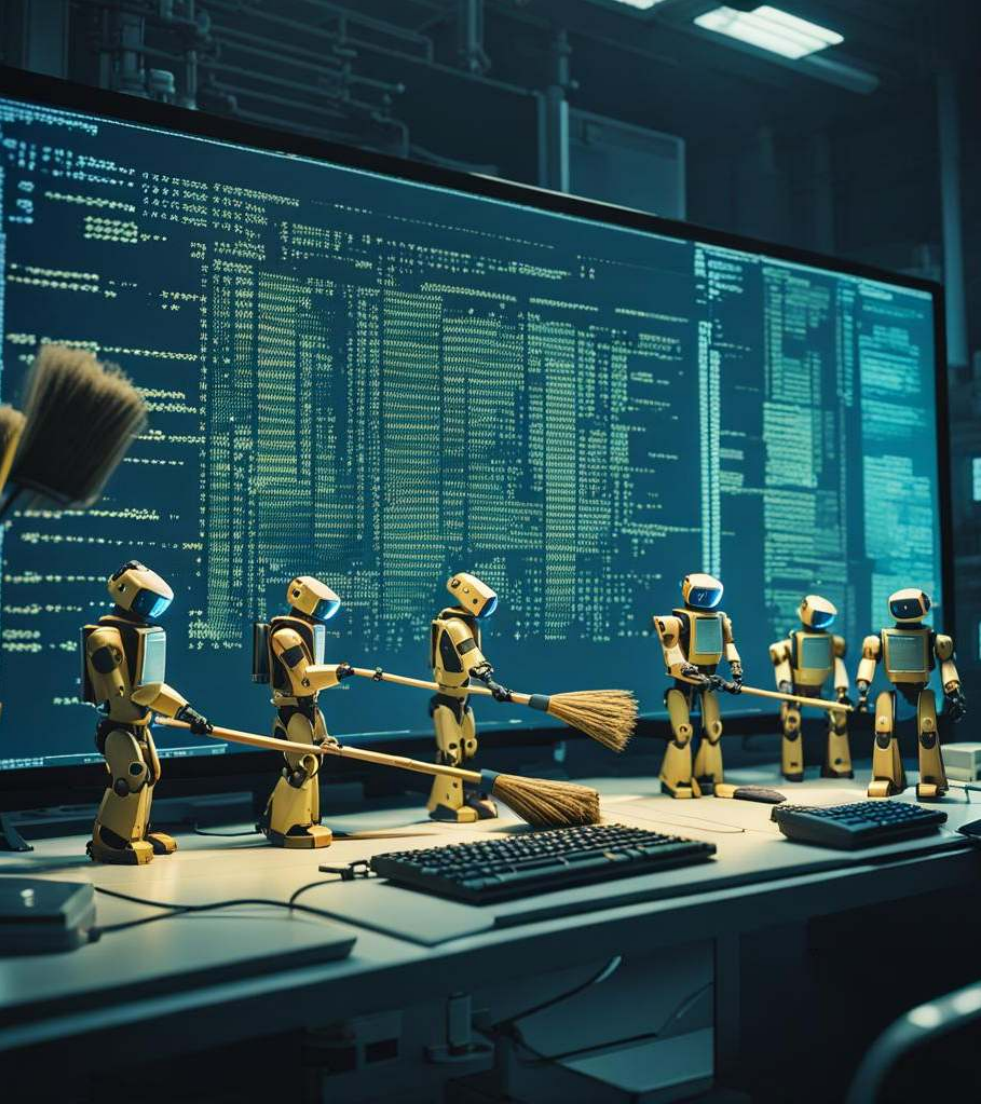
```
1 public static void main(String[] args) {
2     HashMap<String, Integer> params = new HashMap<>();
3     params.put("width", 10);
4     params.put("height", 20);
5     calculateArea(params);
6 }
7 static void calculateArea(HashMap<String, Integer> params) {
8     int area = params.get("width") * params.get("height");
9     System.out.println("Area: " + area);
10 }
```

Explicit Parameters

```
public static void main(String[] args) {  
    HashMap<String, Integer> params = new HashMap<>();  
    params.put("width", 10);  
    params.put("height", 20);  
    calculateArea(params);  
}  
static void calculateArea(HashMap<String, Integer> params) {  
    int area = params.get("width") * params.get("height");  
    System.out.println("Area: " + area);  
}
```



```
public static void main(String[] args) {  
    HashMap<String, Integer> params = new HashMap<>();  
    params.put("width", 10);  
    params.put("height", 20);  
    calculateArea(params.get("width"), params.get("height"));  
}  
static void calculateArea(int width, int height) {  
    int area = width * height;  
    System.out.println("Area: " + area);  
}
```



More Tidying

- Chunk Statements
- Extract Helpers
- One Pile
- Explaining Comments
- Delete Redundant Comments
- even more...



Tidying!

▼ Part I. Tidying

Chapter 1. Guard Clauses

Chapter 2. Dead Code

Chapter 3. Normalize Symmetries

Chapter 4. New Interface, Old Implementation

Chapter 5. Reading Order

Chapter 6. Cohesion Order

Chapter 7. Move Declaration and Initialization Together

Chapter 8. Explaining Variables

Chapter 9. Explaining Constants

Chapter 10. Explicit Parameters

Chapter 11. Chunk Statements

Chapter 12. Extract Helper

Chapter 13. One Pile

Chapter 14. Explaining Comments

Chapter 15. Delete Redundant Comments